Dealing with Exceptions in Scala

Scala and Exceptions

- Scala designed to run on the JVM
  - Exceptions are a basic feature of the JVM
  - Many Java methods will throw exceptions

- Scala does not support checked exceptions
  - Handling is not enforced

- try { … } catch { … } syntax supported
  - But with some differences

```
scala> val s = "123"
s: String = 123

scala> try {
     |    s.toInt
     | } catch {
     |    case e: Exception => println("oops")
     | }
res0: AnyVal = 123
```

Scala and Exceptions

- try { … } catch { … } is an expression
    - Yields a value, but what is the type of this value?

```
scala> val i = try {
     |   s.toInt
     | } catch {
     |   case e: Exception => println("oops")
     | }
i: AnyVal = 123
```

- Could return special value (e.g. -1)
    - Defeats the point of exceptions!!

```
scala> val s = "Foobar"
s: String = xxx

scala> val i = try {
     |   s.toInt
     | } catch {
     |   case e: Exception => println("oops")
     | }
oops
i: AnyVal = ()
```

Using Option[T]

- Encapsulate result in Option[T] type

```
scala> val s = "123"
s: String = 123

scala> val i = try {
     |   Some(s.toInt)
     | } catch {
     |   case e: Exception => None
     | }
i: Option[Int] = Some(123)
```

```
scala> val s = "Foobar"
s: String = Foobar

scala> val i = try {
     |   Some(s.toInt)
     | } catch {
     |   case e: Exception => None
     | }
i: Option[Int] = None
```

- Further processing of result can take place

```
scala> i map ( _ + 4 )
res1: Option[Int] = Some(127)
```

```
scala> i map ( _ + 4 )
res2: Option[Int] = None
```

Using Option[T]

- Use of Option[T] may lead to loss of information
    - Details of exceptions

```
scala> val data = List ( "0", "1", "blah", "2" )
data: List[String] = List(0, 1, blah, 2)

scala> for (
     |   a <- data;
     |   b <- try { Some(a.toInt) }
     |        catch { case ex: Exception => None };
     |   c <- try { Some( 12 / b ) }
     |        catch { case ex2: Exception => None } )
     | yield ( a, c )
res3: List[(String, Int)] = List( (1,12), (2,6) )
```

"blah" fails here

"0" fails here

The Try[T] Type

- Sealed ADT like Option[T]
    - Captures details of non-fatal exceptions
    - Serious faults (e.g. Errors) will still be thrown
    - scala.util.control.NonFatal used to determine if Fatal or Nonfatal

```
scala> import scala.util.{Try, Success, Failure}
import scala.util.{Try, Success, Failure}

scala> Try("123".toInt)
res4: scala.util.Try[Int] = Success(123)

scala> Try("Foobar".toInt)
res5: scala.util.Try[Int] = Failure(java.lang.NumberFormatException:
                                                   For input string: "Foobar")
```

Working with Try[T]

- Higher order functions can be used

```
scala> val result = Try { "123".toInt } map { n => n * 2 }
result: scala.util.Try[Int] = Success(246)

scala> val result = Try { "blah".toInt } map { n => n * 2 }
result: scala.util.Try[Int] = Failure(java.lang.NumberFormatException:
                                                        For input string: "blah")

scala> val result = Try { "0".toInt } flatMap { i => Try { 12 / i } }
result: scala.util.Try[Int] = Failure(java.lang.ArithmeticException: / by zero)
```

Working with Try[T]

- Use get method to retrieve value
    - Throws exception if one exists
- Allows "effect" to be exposed at appropriate stage

```
scala> val result = Try { "123".toInt } map { n => n * 2 } get
result: Int = 246

scala> val result = Try { "blah".toInt } map { n => n * 2 } get
java.lang.NumberFormatException: For input string: "blah"
  at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
  …

scala> val result = Try { "0".toInt } flatMap { i => Try { 12 / i } } get
java.lang.ArithmeticException: / by zero
  at $anonfun$2$$anonfun$apply$1.apply$mcI$sp(<console>:14)
  …
```

Chaining Calls Using Try[T]

- Common Scala idiom
  - Follow the "happy path"
  - Keep track of failure details

```
scala> val dividend = "15"
dividend: String = 15

scala> val divisor = "3"
divisor: String = 3

scala> for ( x <- Try { dividend.toInt };
     |       y <- Try { divisor.toInt } )
     | yield ( x / y )
res7: scala.util.Try[Int] = Success(5)
```

```
scala> val dividend = "foobar"
dividend: String = foobar

scala> val divisor = "3"
divisor: String = 3

scala> for ( x <- Try { dividend.toInt };
     |       y <- Try { divisor.toInt } )
     | yield ( x / y )
res8: scala.util.Try[Int] = Failure(java.lang.NumberFormatException:
                                           For input string: "foobar")
```